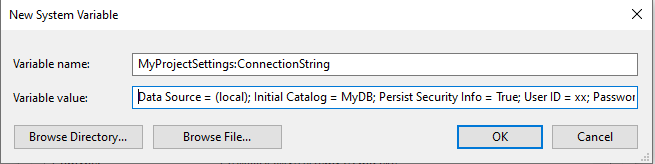
**Dynamic Application Configuration**

One of the problems that we face when moving code from one environment to another is configuration. On a student’s computer a project’s configuration settings are specific to their home environment. When this project is downloaded to a teacher’s computer, many of these configuration settings will have to change. A Database connection string is just one such example. To that end, I have an assembly that contains code to fix this problem.

The assembly you will need is called “ConfigurationAssistant”. It is designed to load configuration settings first from appsettings.json, and if configured to do so, override these settings with user secrets and/or environment settings. Which of these settings is used for your final configuration depends on your initial settings in appsettings.json. There are 3 entries that must appear in your appsettings.json file.

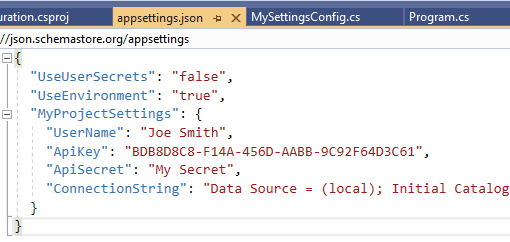
**UseUserSecrets**: If true, then it is expected that the project has created a user secret (using “Manage User Secrets”) project menu. The contents of the secrets.json file should be EXACTLY the same as the “MyProjectSettings” section of appsettings.json. The values in secrets.json will override the values found in appsettings.json.

**UseEnvironment**: If true, then any value from MyProjectSettings that you want to override should have an environment variable appropriately named. Naming for environment variables must include all the keys of the property that you are trying to override. For example, to override the “ConnectionString” value specified in MyProjectSettings, you would create the following SYSTEM environment variable:

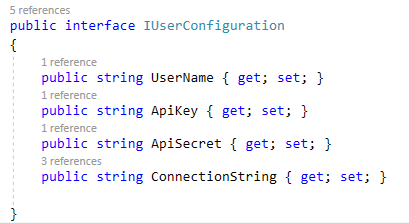


**MyProjectSettings**:

The properties you place into the MyProjectSettings section of appsettings.json are strictly up to you. You can customize this any way you want. The code in ConfigurationAssistant will bind every property in the MyProjectSettings section into a property in an IUserConfiguration interface. Therefore, you must ensure that that Interface is structured EXACTLY like the MyProjectSettings section.

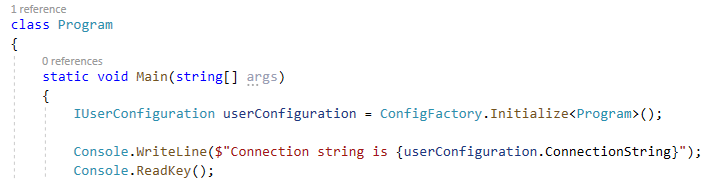


I have placed a few commonly needed property values in this section. If you don’t need them, you can leave then blank. I’ve found that almost every application needs a database connection string, and any application connecting to a web service needs some kind of login information from the client. With this in mind, the initial interface provided for you to access your configuration looks like this:



Remember that you can modify this interface to include ANYTHING you want. This is just a starting point.

The sample application “CustomConfiguration” allows you to play around with the settings to see how it works. Here’s the code needed to retrieve the “ConnectionString” property:



You call “ConfigFactory.Initialize<XXX>()” to retrieve the IUserConfiguration interface through which you can access any property that you had in the MyProjectSettings. XXX specifies any class that resides in the assembly where the user secret was created. The class XXX tells the code which .csproj file to inspect to find the user secret. That .csproj file will contain a section “<UserSecretsId>” continating the user secret. If you are not implementing user secrets, the type you specify here will be ignored.

With the appsettings.json file configured as in the example, if you run the test program now, you’ll see the connection string:

"Data Source = (local); Initial Catalog = AdventureWorks2017; Persist Security Info = True; User ID = sa; Password=ZZZ;"

However, if you create a SYSTEM environment variable as in the example above, then when you run the test you’ll see:

“Data Source = (local); Initial Catalog = MyDB; Persist Security Info = True; User ID = xx; Password=abc;”

If you construct your application to read configuration information in this way, then you won’t have a problem overriding configuration values from projects that you clone from GitGub or any other location.

One final note, if you create or change an environment variable while you have Visual Studio running, then you’ll need to close and reopen VS in order for it to notice the change. Unfortunately, VS does not read environment variables on the fly. It caches them when it loads and keeps those values until it is restarted again.